Créer une API Web C# avec Entity Framework et MySQL - Projet Restaurant

Guide complet pour créer une API web REST avec C#, Entity Framework Core et MySQL dans Visual Studio. Apprenez à configurer EF, créer des modèles, gérer les migrations et générer des controllers automatiquement.

Programmation Laravel 30 min de lecture Niveau Intermédiaire

Document généré le 11/07/2026 à 20h40 · nouvy.fr/wiki/api-web-csharp-entity-framework-mysql-restaurant

Sommaire

35 section(s) · 30 min de lecture

🔧 Résumé des commandes importantes

↳ Installation des packages

↳ Migrations

↳ Génération de controllers

🚀 Prochaines étapes

⚠️ Points d'attention

🎯 Conclusion

Ce guide vous apprendra à créer une **API web REST** complète avec **C#**, **Entity Framework Core** et **MySQL** dans **Visual Studio**. Nous utiliserons un projet de gestion de restaurant comme exemple pratique.

□ Objectifs

À la fin de ce guide, vous serez capable de :

- □ Créer un projet API web ASP.NET Core dans Visual Studio
- □ Configurer Entity Framework Core avec MySQL
- □ Créer des modèles de données et définir les relations
- □ Générer et appliquer des migrations de base de données
- □ Créer automatiquement des controllers REST avec le code generator
- □ Tester votre API avec Swagger

□ Prérequis

- **Visual Studio** (2019 ou supérieur) avec le workload ASP.NET Core
- **MySQL** installé et configuré sur votre machine
- **.NET SDK 8.0** installé sur votre machine
- Connaissances de base en C# et SQL

Étape 1 : Création du projet dans Visual Studio

1. Ouvrez **Visual Studio**

2. Cliquez sur **"Créer un nouveau projet"**

3. Sélectionnez le modèle **"API Web ASP.NET Core"**

4. Configurez votre projet :
    - **Nom du projet** : `RestaurantAPI` (ou le nom de votre choix)
    - **Framework** : .NET 8.0
    - **Authentification** : Aucune (pour simplifier)
    - **Activer OpenAPI** : Oui (pour Swagger)

5. Cliquez sur **"Créer"**

Votre projet est maintenant créé avec la structure de base d'une API ASP.NET Core.

Étape 2 : Installation des packages NuGet

Ouvrez PowerShell dans le répertoire de votre projet et exécutez les commandes suivantes :

Installation de l'outil dotnet-ef en ligne de commande

Cette commande installe l'outil EF Core CLI globalement sur votre machine. Vous n'aurez besoin de le faire qu'une seule fois :

```
dotnet tool install --global dotnet-ef --version 9.0.9
```

Copier

Installation des packages NuGet

Installez les packages suivants avec les versions exactes spécifiées :

```
dotnet add package Microsoft.EntityFrameworkCore.Design --version 9.0.9
dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 9.0.9
dotnet add package Microsoft.EntityFrameworkCore.Tools --version 9.0.9
dotnet add package Pomelo.EntityFrameworkCore.MySql --version 9.0.0
```

Copier

> ***Résolution d'erreur*** *: Si vous rencontrez l'erreur suivante lors de l'installation de* `dotnet-ef` *:*
>
> ```
> Échec de la mise à jour de l'outil 'dotnet-ef' pour la ou les raisons suivantes :
> Le fichier de paramètres dans le package NuGet de l'outil n'est pas valide :
> Le fichier de paramètres 'DotnetToolSettings.xml' est introuvable dans le package.
> Impossible d'installer l'outil 'dotnet-ef'. Contactez le créateur de l'outil pour
> obtenir de l'aide.
> ```
>
> Copier
>
> ***Solution*** *: Désinstallez d'abord l'outil existant, puis réinstallez-le :*
>
> ```
> dotnet tool uninstall --global dotnet-ef
> dotnet tool install --global dotnet-ef --version 9.0.9
> ```
>
> Copier

Étape 3 : Création des modèles de données

Nous allons créer les modèles pour notre application de restaurant. Créez un dossier **`Models`** dans votre projet (clic droit sur le projet > `Ajouter` > `Nouveau dossier`).

Modèle Article

Créez une nouvelle classe C# dans le dossier `Models` et nommez-la `Article.cs` :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RestaurantAPI.Models
{
    public class Article
    {
        public int Id { get; set; }
        public string Nom { get; set; }
        public float PrixHT { get; set; }
        public float PrixTTC { get; set; }
        public float TauxTva { get; set; }

        // Relations de navigation
        public List<Menu> Menus { get; set; }
        public List<Commande> Commandes { get; set; }
    }
}
```

Copier

Modèle Commande

Créez `Commande.cs` :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RestaurantAPI.Models
{
    public class Commande
    {
        public int Id { get; set; }
        public string Numero { get; set; }
        public float TotalHT { get; set; }
        public float TotalTTC { get; set; }

        // Relations de navigation
        public List<Menu> Menus { get; set; }
        public List<Article> Articles { get; set; }

        // Clés étrangères
        public int TableId { get; set; }
        public Table Table { get; set; }

        public int UtilisateurId { get; set; }
        public Utilisateur Utilisateur { get; set; }
    }
}
```

Copier

Modèle Menu

Créez `Menu.cs` :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RestaurantAPI.Models
{
    public class Menu
    {
        public int Id { get; set; }
        public string Nom { get; set; }
        public float PrixHT { get; set; }
        public float PrixTTC { get; set; }
        public float TauxTva { get; set; }

        // Relations de navigation
        public List<Article> Articles { get; set; }
        public List<Commande> Commandes { get; set; }
    }
}
```

Copier

Modèle Table

Créez `Table.cs` :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RestaurantAPI.Models
{
    public class Table
    {
        public int Id { get; set; }
        public int Numero { get; set; }
        public int Places { get; set; }

        // Relations de navigation
        public List<Commande> Commandes { get; set; }
    }
}
```

Copier

Modèle Utilisateur

Créez `Utilisateur.cs` :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RestaurantAPI.Models
{
    public class Utilisateur
    {
        public int Id { get; set; }
        public string Nom { get; set; }
        public string Prenom { get; set; }
        public string Email { get; set; }
        public string Password { get; set; }

        // Relations de navigation
        public List<Commande> Commandes { get; set; }
    }
}
```

Copier

🔗 Notes sur les relations

Les modèles définissent les relations entre les tables :

- **Article** ↔ **Menu** : Relation many-to-many (un article peut être dans plusieurs menus, un menu contient plusieurs articles)
- **Article** ↔ **Commande** : Relation many-to-many (une commande peut contenir plusieurs articles)
- **Menu** ↔ **Commande** : Relation many-to-many (une commande peut contenir plusieurs menus)
- **Commande** → **Table** : Relation many-to-one (plusieurs commandes peuvent être associées à une table)
- **Commande** → **Utilisateur** : Relation many-to-one (plusieurs commandes peuvent être créées par un utilisateur)

Étape 4 : Création du contexte Entity Framework

Créez une nouvelle classe `RestaurantContext.cs` à la racine de votre projet (ou dans un dossier `Data`) :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using RestaurantAPI.Models;

namespace RestaurantAPI
{
    public class RestaurantContext : DbContext
    {
        public RestaurantContext(DbContextOptions<RestaurantContext> options)
            : base(options)
        {
        }

        public DbSet<Article> Articles { get; set; }
        public DbSet<Menu> Menus { get; set; }
        public DbSet<Table> Tables { get; set; }
        public DbSet<Commande> Commandes { get; set; }
        public DbSet<Utilisateur> Utilisateurs { get; set; }
    }
}
```

Copier

🔧 Configuration de la chaîne de connexion

Modifiez la chaîne de connexion selon votre configuration MySQL :

- **Server** : Adresse du serveur MySQL (généralement `localhost`)
- **Database** : Nom de la base de données (sera créée automatiquement lors de la migration)
- **User** : Nom d'utilisateur MySQL
- **Password** : Mot de passe MySQL

> ⚠ ***Important*** *: En production, utilisez la configuration dans* `appsettings.json` *et les variables d'environnement pour sécuriser vos identifiants.*

Étape 5 : Configuration du DbContext dans Program.cs

Pour utiliser Entity Framework dans votre API, vous devez enregistrer le contexte dans le conteneur d'injection de dépendances. Modifiez votre fichier `Program.cs` :

```
using Microsoft.EntityFrameworkCore;
using RestaurantAPI;

var builder = WebApplication.CreateBuilder(args);

// Ajouter les services au conteneur
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Enregistrer le DbContext
builder.Services.AddDbContext<RestaurantContext>(options =>
    options.UseMySql(
        builder.Configuration.GetConnectionString("DefaultConnection")
        ?? "Server=localhost;Database=restaurant_api;User=root;Password=mysql;",
ServerVersion.AutoDetect("Server=localhost;Database=restaurant_api;User=root;Password=mysql;
")
    ));

var app = builder.Build();

// Configurer le pipeline HTTP
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();

app.Run();
```

Copier

Configuration dans appsettings.json

Ajoutez la chaîne de connexion dans `appsettings.json` :

```
{
  "ConnectionStrings": {
    "DefaultConnection":
"Server=localhost;Database=restaurant_api;User=root;Password=mysql;"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Copier

Étape 6 : Création et application des migrations

Une fois tous les modèles créés, nous allons générer la base de données à partir de nos modèles.

Créer la migration initiale

Ouvrez la **Console du Gestionnaire de packages** et exécutez :

```
dotnet ef migrations add InitialCreate
```

Copier

Cette commande crée un dossier `Migrations` dans votre projet contenant les fichiers de migration nécessaires pour créer votre base de données.

Appliquer la migration à la base de données

Exécutez la commande suivante pour créer la base de données et les tables :

```
dotnet ef database update
```

Copier

> ***Résultat attendu** : La base de données `restaurant_api` est créée avec toutes les tables (Articles, Menus, Tables, Commandes, Utilisateurs) et les relations appropriées.*

Étape 7 : Génération automatique des controllers

Pour générer automatiquement les controllers REST avec toutes les opérations CRUD, nous utilisons le **scaffolding** d'ASP.NET Core.

Installation des outils nécessaires

Installez d'abord l'outil de génération de code :

```
dotnet tool install -g dotnet-aspnet-codegenerator
```

Copier

Les packages suivants doivent déjà être installés (voir Étape 2) :

- `Microsoft.VisualStudio.Web.CodeGeneration.Design` (version 9.0.0)
- `Microsoft.EntityFrameworkCore.SqlServer` (version 9.0.9)

Si vous ne les avez pas encore installés, exécutez :

```
dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design --version 9.0.0
dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 9.0.9
```

Copier

Génération d'un controller

Pour générer un controller pour l'entité `Commande`, ouvrez la **Console du Gestionnaire de packages** et exécutez :

```
dotnet aspnet-codegenerator controller -name CommandeController -async -api -m Commande -dc
RestaurantContext -outDir Controllers
```

Copier

Paramètres de la commande :

- `-name CommandeController` : Nom du controller à créer
- `-async` : Génère des méthodes asynchrones
- `-api` : Génère un controller API (sans vues)
- `-m Commande` : Modèle à utiliser
- `-dc RestaurantContext` : Contexte de données
- `-outDir Controllers` : Dossier de sortie

Générer les autres controllers

Répétez la commande pour chaque entité :

```
dotnet aspnet-codegenerator controller -name ArticleController -async -api -m Article -dc
RestaurantContext -outDir Controllers

dotnet aspnet-codegenerator controller -name MenuController -async -api -m Menu -dc
RestaurantContext -outDir Controllers

dotnet aspnet-codegenerator controller -name TableController -async -api -m Table -dc
RestaurantContext -outDir Controllers

dotnet aspnet-codegenerator controller -name UtilisateurController -async -api -m
Utilisateur -dc RestaurantContext -outDir Controllers
```

Copier

Étape 8 : Test de l'API avec Swagger

Lancement de l'application

1. Appuyez sur **F5** ou cliquez sur **"Exécuter"** dans Visual Studio
2. L'application démarre et ouvre automatiquement Swagger dans votre navigateur
3. L'URL sera similaire à : `https://localhost:7115/swagger/index.html` (le port peut varier selon votre configuration)

Routes générées automatiquement

Pour chaque controller, les routes suivantes sont créées :

Routes Commande

- **GET** `/api/Commande` - Récupère toutes les commandes
- **POST** `/api/Commande` - Crée une nouvelle commande
- **GET** `/api/Commande/{id}` - Récupère une commande par ID
- **PUT** `/api/Commande/{id}` - Met à jour une commande
- **DELETE** `/api/Commande/{id}` - Supprime une commande

Les mêmes routes sont disponibles pour **Article**, **Menu**, **Table** et **Utilisateur**.

Tester l'API

1. Dans Swagger, cliquez sur **"GET /api/Commande"** pour voir la liste des commandes
2. Cliquez sur **"Try it out"** puis sur **"Execute"**
3. Vous devriez recevoir une réponse JSON (probablement vide si aucune donnée n'a été ajoutée)

□ Résumé des commandes importantes

Installation des packages

```
# Installation de l'outil dotnet-ef
dotnet tool install --global dotnet-ef --version 9.0.9

# Installation des packages NuGet
dotnet add package Microsoft.EntityFrameworkCore.Design --version 9.0.9
dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 9.0.9
dotnet add package Microsoft.EntityFrameworkCore.Tools --version 9.0.9
dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design --version 9.0.0
dotnet add package Pomelo.EntityFrameworkCore.MySql --version 9.0.0
```

Copier

> ***Note** : Si vous rencontrez une erreur lors de l'installation de `dotnet-ef`, désinstallez d'abord l'outil existant puis réinstallez-le :*
>
> ```
> dotnet tool uninstall --global dotnet-ef
> dotnet tool install --global dotnet-ef --version 9.0.9
> ```
>
> Copier

Migrations

```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

Copier

Génération de controllers

```
dotnet tool install -g dotnet-aspnet-codegenerator
dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design --version 9.0.0
dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 9.0.9
dotnet aspnet-codegenerator controller -name [NomController] -async -api -m [NomModele] -dc
RestaurantContext -outDir Controllers
```

Copier

□ Prochaines étapes

Maintenant que votre API de base est fonctionnelle, vous pouvez :

1. **Configurer les relations many-to-many** : Ajouter les tables de jointure pour les relations Article-Menu, Article-Commande, etc.
2. **Ajouter la validation** : Utiliser les Data Annotations pour valider les données
3. **Implémenter l'authentification** : Ajouter JWT ou Identity pour sécuriser l'API
4. **Ajouter la pagination** : Pour les listes de données importantes
5. **Implémenter le filtrage et la recherche** : Permettre de filtrer les résultats
6. **Ajouter la documentation XML** : Améliorer la documentation Swagger
7. **Configurer CORS** : Pour permettre les appels depuis un frontend
8. **Ajouter les tests unitaires** : Tester vos controllers et services

---

⚠ Points d'attention

- **Sécurité** : Ne stockez jamais les mots de passe en clair. Utilisez le hachage (BCrypt, Argon2, etc.)
- **Gestion des erreurs** : Implémentez une gestion d'erreurs globale avec des middlewares
- **Performance** : Utilisez `AsNoTracking()` pour les requêtes en lecture seule
- **Chaîne de connexion** : Utilisez les variables d'environnement en production
- **Relations** : Configurez correctement les relations many-to-many avec `OnModelCreating`

□ Conclusion

Vous avez maintenant créé une API web REST complète avec C#, Entity Framework Core et

MySQL ! Votre API est prête à être utilisée et peut être étendue selon vos besoins spécifiques.

Pour toute question ou problème, consultez la documentation officielle :

- Entity Framework Core
- Pomelo.EntityFrameworkCore.MySql
- ASP.NET Core Web API